

1      **DISK DRIVE FOR RECEIVING SETUP DATA IN A SELF MONITORING**  
2      **ANALYSIS AND REPORTING TECHNOLOGY (SMART) COMMAND**

3

4      **BACKGROUND OF THE INVENTION**

5      **Field of the Invention**

6            The present invention relates to disk drives. More particularly, the present invention  
7      relates to a disk drive for receiving setup data in a self monitoring analysis and reporting  
8      technology (SMART) command.

9      **Description of the Prior Art**

10         Conventional disk drives comprise a sector of configuration parameters used to configure  
11      the operating mode when powered on. The configuration parameters are used to configure, for  
12      example, the read/write channel, the cache system, and the error recovery system. The  
13      configuration parameters are generally inaccessible from a host computer due to the limitations of  
14      its operating system, such as a Microsoft Windows operating system and associated drivers.

15         A utility has been disclosed for configuring a write-verify error recovery operation from  
16      the host computer. However, this utility can only be executed after the host computer is “clean”  
17      booted from a floppy disk so that the Microsoft Windows drivers are not installed. After running  
18      the utility to configure the write-verify operation, the host computer is rebooted into the  
19      Microsoft Windows operating system to resume normal operation.

20         Rebooting the host computer from a floppy in order to reconfigure a disk drive is  
21      undesirable since it precludes running the configuration utility from the more user friendly and  
22      familiar Microsoft Windows operating environment. In addition, it may be desirable to run the  
23      configuration utility from an Internet web page using an Internet browser program running under  
24      a Microsoft Windows operating environment. Still further, it may be desirable to reconfigure the  
25      disk drive on-the-fly relative to the type of application program running, or the type of data being  
26      manipulated. For example, it may be desirable to configure the error recovery system less  
27      stringently when storing Internet web pages in a browser’s cache as opposed to storing more

1 critical word processing or accounting documents.

2 There is, therefore, a need to reconfigure a disk drive in the field from a Microsoft  
3 Windows operating system without having to reboot the host computer.

4 **SUMMARY OF THE INVENTION**

5 The present invention may be regarded as a disk drive connectable to a host computer  
6 executing a computer program for sending a Self Monitoring Analysis and Reporting Technology  
7 (SMART) command to the disk drive. The disk drive comprises a disk, a head actuated radially  
8 over the disk, an error recovery system for detecting and correcting errors in user data read from  
9 the disk, and a cache system for caching user data received from the host computer and user data  
10 read from the disk. The disk drive further comprises a plurality of configuration parameters  
11 stored in a non-volatile manner for configuring at least one system when the disk drive is powered  
12 on, the at least one system selected from the group consisting of the cache system and the error  
13 recovery system. The disk drive comprises an interface for receiving the SMART command from  
14 the host computer. The SMART command comprises a command code comprising a  
15 predetermined value for identifying the command code as a SMART command and a sub  
16 command comprising one of a plurality of predetermine values identifying one of a plurality of  
17 SMART commands selected from the group consisting of enabling SMART diagnostics, reading  
18 diagnostic data, and transmitting setup data to the disk drive. The SMART command further  
19 comprises setup data for modifying the configuration parameters.

20 In one embodiment the disk drive further comprises a volatile semiconductor memory.  
21 When the disk drive is powered on, the configuration parameters are copied to the volatile  
22 semiconductor memory, and the setup data is used to modify the configuration parameters stored  
23 in the volatile semiconductor memory in order to configure the at least one system on-the-fly.

24 In one embodiment the error recovery system comprises a plurality of retry procedures  
25 responsive to the configuration parameters. In another embodiment, the error recovery system  
26 comprises an error correction code (ECC) system responsive to the configuration parameters. In  
27 yet another embodiment, the ECC system comprises a parity sector mode wherein the ECC

1 system writes parity sectors to the disk, and the configuration parameters enable the parity sector  
2 mode.

3 In one embodiment, the disk drive comprises a write-verify system for verifying a write  
4 operation by verifying recoverability of written data, wherein the at least one system configured  
5 using the configuration parameters includes the write-verify system. In one embodiment, the  
6 configuration parameters enable the write-verify system.

7 In one embodiment the cache system comprises a semiconductor memory, and the cache  
8 system reserves a block of the semiconductor memory for caching data read from the disk during  
9 a read operation. The configuration parameters configure when the cache system releases the  
10 reserved block of semiconductor memory. In another embodiment, the configuration parameters  
11 configure a number of blocks reserved in the semiconductor memory for caching write data  
12 received from the host computer. In yet another embodiment, the disk comprises a plurality of  
13 tracks, where each track comprises a plurality of sectors. The configuration parameters configure  
14 a number of sectors read into the semiconductor memory during a read operation following a  
15 target sector of the read operation. In another embodiment, the configuration parameters  
16 configure a number of sectors read into the semiconductor memory during a read operation  
17 preceding a target sector of the read operation.

18 In one embodiment, the computer program executed by the host computer comprises a  
19 graphical user interface for generating the setup data in response to user input. In another  
20 embodiment, the computer program executed by the host computer is a user application program  
21 which generates the setup data independent of user input.

22 In one embodiment, the setup data is received over the Internet. An Internet web server  
23 downloads a web page to a computer connected to the disk drive. The computer displays the web  
24 page in an Internet browser program. In response to user input, the web server transmits a  
25 SMART command to the disk drive via the Internet and the computer connected to the disk drive.

26 The present invention may also be regarded as a computer program embodied on a  
27 computer readable storage medium for use in a host computer. The computer program for

1 configuring a disk drive by transmitting setup data in a Self Monitoring Analysis and Reporting  
2 Technology (SMART) command to the disk drive. The SMART command comprises a  
3 command code, a sub command, and a buffer. The disk drive comprises a disk, a head actuated  
4 radially over the disk, an error recovery system for detecting and correcting errors in user data  
5 read from the disk, and a cache system for caching user data received from the host computer and  
6 user data read from the disk. The disk drive further comprises a plurality of configuration  
7 parameters stored in a non-volatile manner for configuring at least one system when the disk drive  
8 is powered on, the at least one system selected from the group consisting of the cache system and  
9 the error recovery system. The computer program comprises code segments for assigning a value  
10 to the command code identifying the command code as a SMART command, assigning a value to  
11 the sub command identifying the sub command as a disk drive setup command, and assigning  
12 setup data to the buffer, the setup data for modifying the configuration parameters of the disk  
13 drive. The computer program further comprises a code segment for transmitting the SMART  
14 command to the disk drive.

#### **BRIEF DESCRIPTION OF THE DRAWINGS**

16 FIG. 1 shows a disk drive according to an embodiment of the present invention for  
17 receiving a SMART command from a host computer, the SMART command comprising setup  
18 data for modifying the configuration parameters of the disk drive.

19 FIG. 2A shows the parameters associated with the function called from a Microsoft  
20 Windows operating system in order to invoke the SMART driver for communicating the setup  
21 data to the disk drive of FIG. 1.

22 FIG. 2B shows the format of a data structure used to send parameters to the disk drive  
23 using the procedure of FIG. 2A.

24 FIG. 2C shows the format of a data structure used to receive parameters from the disk  
25 drive using the procedure of FIG. 2A.

26 FIG. 2D shows the format of a data structure associated with an IDE command.

27 FIG. 3 shows a disk drive configuration utility executed by the host computer according to

1 an embodiment of the present invention.

2 FIG. 4 shows an embodiment wherein the disk drive configuration utility is executed by a  
3 web server via the Internet.

4 **DESCRIPTION OF THE PREFERRED EMBODIMENTS**

5 FIG. 1 shows a disk drive 2 according to an embodiment of the present invention  
6 connectable to a host computer 4 executing a computer program for sending a Self Monitoring  
7 Analysis and Reporting Technology (SMART) command 6 to the disk drive 2. The disk drive 2  
8 comprises a disk 8, a head 10 actuated radially over the disk 8, an error recovery system 12 for  
9 detecting and correcting errors in user data read from the disk 8, and a cache system 14 for  
10 caching user data received from the host computer 4 and user data read from the disk 8. The disk  
11 drive 2 further comprises a plurality of configuration parameters stored in a non-volatile manner  
12 for configuring at least one system when the disk drive 2 is powered on, the at least one system  
13 selected from the group consisting of the cache system 14 and the error recovery system 12. The  
14 disk drive 2 comprises an interface 16 for receiving the SMART command 6 from the host  
15 computer 4. The SMART 6 command comprises a command code 18 comprising a  
16 predetermined value for identifying the command code as a SMART command and a sub  
17 command 20 comprising one of a plurality of predetermine values identifying one of a plurality of  
18 SMART commands selected from the group consisting of enabling SMART diagnostics, reading  
19 diagnostic data, and transmitting setup data to the disk drive 2. The SMART command 6 further  
20 comprises setup data 22 for modifying the configuration parameters.

21 In the embodiment of FIG. 1 the disk drive 2 comprises a disk controller 24 for  
22 implementing the various disk drive systems, including the error recovery system 12 and the cache  
23 system 14. A semiconductor memory 26 stores the configuration parameters for access by the  
24 disk controller 24. In one embodiment, the configuration parameters are stored in a reserved  
25 sector on the disk 8. When powered on, the disk drive 2 reads the configuration parameters from  
26 the disk 8 and stores the configuration parameters in the semiconductor memory 26. In one  
27 embodiment, the setup data 22 received in the SMART command modifies both the configuration

1 parameters stored on the disk 8 as well as the configuration parameters stored in the  
2 semiconductor memory 26 so that the disk drive 2 is configured on-the-fly.

3 The disk drive 2 of FIG. 1 further comprises a spindle motor 28 for rotating the disk 8 and  
4 a voice coil motor (VCM) 30 for actuating the head 10 radially over the disk 8. A servo  
5 controller 32 generates the appropriate control signals applied to the spindle motor 28 and VCM  
6 30 in response to commands received from the disk controller 24. During a write operation the  
7 disk controller 24 transmits user data received from the host computer 4 to a read/write channel  
8 34. The read/write channel 34 performs appropriate encoding of the user data to generate write  
9 data 36 written to the disk 8. The write data 36 modulates the operation of a preamp 38 to  
10 generate a write signal 40<sub>i</sub> applied to the head 10 in order to write magnetic transitions onto the  
11 surface of the disk 8. During a read operation, the head 10 detects the magnetic transitions  
12 representing the recorded data to generate a read signal 42<sub>i</sub> which is amplified by the preamp 38  
13 to generate a read signal 44 applied to the read/write channel 34. The read/write channel 34  
14 demodulates the read signal 44 into user data transmitted to the host computer 4 via the disk  
15 controller 24 after correcting errors using the error recovery system 12. In the embodiment of  
16 FIG. 1 embedded servo data is recorded on the disk 8, demodulated by the read/write channel 34,  
17 and used by the servo controller 32 to position the head 10 over the target data track.

18 The disk drive 2 communicates with the host computer 4 through a communication  
19 protocol referred to as the IDE protocol. The Microsoft Windows operating system comprises a  
20 low level “port” driver which communicates with the disk drive 2 through an IDEREGS data  
21 structure shown in FIG. 2D. There are only a limited number of IDE commands supported by  
22 disk drive manufactures as well as the Microsoft Windows operating systems. The standard IDE  
23 protocol provides no support for modifying the configuration parameters of a disk drive by a  
24 computer program running under a Microsoft Windows operating system. However, the standard IDE  
25 protocol does provide support for the Self Monitoring Analysis and Reporting Technology  
26 (SMART) which allows a computer program running under a Microsoft Windows operating  
27 system to configure a disk drive so that it will accumulate and report diagnostic information. In

1 this manner, the host computer can evaluate the health of the disk drive and report impending  
2 failures to the user so that remedial measures can be taken before data is lost. Further details  
3 regarding the application programming interface (API) for the SMART protocol can be found in  
4 the "Windows 95 and Windows NT SMART IOCTL API Specification" version 01.02 which is  
5 incorporated herein by reference.

6 An IDE command of 0xB0 (inserted into the chComandReg field of the IDEREGS data  
7 structure of FIG. 2D) identifies the IDE command as a SMART command. A predetermined  
8 number of "sub commands" have been defined for the SMART protocol (inserted into the  
9 chFeaturesReg field of the IDEREGS data structure of FIG. 2D), including a sub command for  
10 enabling/disabling the SMART system within the disk drive, as well as a sub command to retrieve  
11 the diagnostic data from the disk drive. In one embodiment of the present invention, at least one  
12 of the sub commands is redefined by the disk drive 2 of FIG. 1 to allow the configuration  
13 parameters within the disk drive 2 to be modified by a computer program running under a  
14 Microsoft Windows operating system.

15 FIG. 2A shows the format of the function called from a computer program running under  
16 a Microsoft Operating system to effectuate the transfer of a SMART command to the disk drive 2  
17 of FIG. 1. This function is installed into a Microsoft Windows operating system through an  
18 appropriate driver, such as the vendor specific driver DFPVSD.VXD. The function  
19 DeviceIoControl has the following input parameters:

20 **HDevice:** Identifies the device. The CreateFile function returns this handle.  
21 **DwIoControlCode:** Specifies the control code for the operation. This value identifies the specific  
22 operation to be performed and the type of device on which the operation is to be performed. The  
23 following values are defined for this driver:

1

Value	Meaning
DFP_GET_VERSION (0x00074080)	Gets the version and revision of the driver.
DFP_SEND_DRIVE_COMMAND (0x0007c084)	Sends a generic command to a drive. Only used to send a command to the drive that sends data or no data is transferred.
DFP_RECEIVE_DRIVE_DATA (0x0007c088)	Sends a command to the drive that returns data

2      **PvInBuffer:** Points to a buffer containing the data required to perform the operation. This  
3      parameter can be NULL if the dwIoControlCode parameter specifies an operation that does not  
4      require input data.

5      **CbInBuffer:** Specifies the size, in bytes, of the lpvInBuffer buffer.

6      **LpvOutBuffer:** Points to a buffer in which the operation's output data is returned. This  
7      parameter can be NULL if the dwIoControlCode parameter specifies an operation that does not  
8      produce output data.

9      **CbOutBuffer:** Specifies the size, in bytes, of the lpvOutBuffer buffer.

10     **LpcbBytesReturned:** Points to a 32-bit variable that receives the size, in bytes, of the data  
11    returned in the lpvOutBuffer buffer.

12     **LpoOverlapped:** Points to an OVERLAPPED structure. This parameter is ignored if the  
13    hDevice handle was opened without specifying the FILE\_FLAG\_OVERLAPPED flag. This  
14    parameter can be NULL if overlapped operation is not desired. OVERLAPPED (asynchronous)  
15    I/O will not be used by this driver, so this parameter should be set to NULL.

16     **Return Value:** If the function succeeds, the return value is TRUE; otherwise, it is FALSE.

17       Before a computer program can call the DeviceIoControl function of FIG. 2A, a handle to  
18    the target driver must be obtained. An OpenSMART procedure is set forth in the attached source

1 code appendix for obtaining a handle to the target driver. The OpenSMART procedure utilizes  
 2 the CreateFile function provided by the Microsoft Windows operating system in order to create  
 3 the handle to the driver. Once the handle is obtained, the target disk drive is enabled to receive  
 4 SMART commands by transmitting the appropriate IDE command. A DoEnableSMART  
 5 procedure is set forth in the source code appendix for enabling the disk drive to receive SMART  
 6 commands.

7 Once the disk drive 2 has been enabled for SMART commands, the setup data for  
 8 modifying the configuration parameters can be transmitted to the disk drive via a computer  
 9 program running under a Microsoft Windows operating system. A DoModifyBlockKeySector is  
 10 set forth in the source code appendix for transmitting the setup data to the disk drive via a  
 11 SMART command. In this embodiment, the setup data is contained within a  
 12 MODIFYBLOCK\_KEYSECTOR data structure. The MODIFYBLOCK\_KEYSECTOR data  
 13 structure comprises 123 MODIFYBLOCK data structures which contain four bytes each as  
 14 defined by the following table:

Field	Bytes	Description
Action code and Word Offset	2	<p>Bits 0 to 9 - Offset from start of configuration data. For the DRAM copy, this the start of the Config. Sector memory table; for the media copy, the first entry ("serial number") after the 24-byte file header.</p> <p>Bits 10 to 12 – Unused (must be zero)</p> <p>Bits 13 to 15 – Modify action code</p> <ul style="list-style-type: none"> <li>0 = No action, unused entry</li> <li>1 = Replace LS byte</li> <li>2 = Replace MS byte</li> <li>3 = Replace word (LS and MS bytes)</li> <li>4 = Set bit (ones in mask are set to one, zeros in mask are unchanged)</li> <li>5 = Clear bit (ones in mask are set to zero, zeros in mask are unchanged)</li> <li>6 to 7 = Undefined</li> </ul>
LS byte	1	Least significant byte of data or mask
MS byte	1	Most significant byte of data or mask

16 The MODIFYBLOCK data structure thus allows individual bits of the configuration parameters

1 to be modified as well as replacing an entire byte.

2 A SENDCMDINPARAMS data structure shown in FIG. 2B is used to send input  
3 parameters to the disk drive 2, and a SENDCMDOUTPARAMS data structure shown in FIG. 2C  
4 is used to receive output parameters from the disk drive. The dwBufferSize field of the  
5 SENDCMDINPARAMS data structure is set to the size of the MODIFYBLOCK\_KEYSECTOR  
6 data structure which is copied into the chBuffer[1] field. The irDriveRegs are configured  
7 appropriately, including to set the SMART sub command field chFeaturesReg to a value  
8 kchWriteLoggingSectorFeaturesRegister which in this embodiment is defined as 0xD6. The  
9 DeviceIoControl function of FIG. 2A is then called with the SENDCMDINARAMS data  
10 structure, including the MODIFYBLOCK\_KEYSECTOR data structure which the disk drive 2  
11 uses to modify the configuration parameters.

12 In one embodiment the error recovery system 12 comprises a plurality of retry procedures  
13 responsive to the configuration parameters. The computer program may configure the retry  
14 procedures in order to optimize the acceptable latency. For example, the computer program may  
15 configure the number of retries to perform before aborting a read operation.

16 In another embodiment, the error recovery system 12 comprises an error correction code  
17 (ECC) system responsive to the configuration parameters. The ECC system implements any  
18 suitable ECC code, such as the well known Reed-Solomon ECC code, wherein a number of  
19 redundancy bytes are generated for each data sector and stored with each data sector. The  
20 computer program may disable the retry procedures and/or the ECC code for audio/visual data in  
21 order to prevent the associated latency from interrupting the continuous transfer of a data stream.

22 In yet another embodiment, the ECC system comprises a parity sector mode wherein the  
23 ECC system writes parity sectors to the disk 2. A parity sector is generated by computing a  
24 parity over a predetermined number of data sectors and storing the parity sector with the data  
25 sectors. In this manner, if one of the data sectors is unrecoverable using the ECC code it can be  
26 reconstructed by combining the parity sector with the other data sectors. The computer program  
27 can enable the parity sector mode in order to increase the redundancy and associated protection

1 for more critical data, such as accounting or word processing data.

2 In one embodiment, the disk drive 2 comprises a write-verify system for verifying a write  
3 operation by verifying recoverability of written data. For example, the disk drive may attempt to  
4 read a recently written data sector to verify the recoverability of the data sector before deleting  
5 the write data from the semiconductor memory 26. If the data sector is unrecoverable, the disk  
6 drive 2 may respond by rewriting the data sector, or relocating the data sector and marking the  
7 unrecoverable data sector as a defective sector. In one embodiment, the configuration parameters  
8 enable/disable the write-verify system to configure the level of protection desired relative to the  
9 associated latency.

10 In one embodiment the cache system 14 reserves a block of the semiconductor memory 26  
11 for caching data read from the disk 8 during a read operation. The configuration parameters  
12 configure when the cache system 14 releases the reserved block of semiconductor memory 26. In  
13 another embodiment, the configuration parameters configure a number of blocks reserved in the  
14 semiconductor memory 26 for caching write data received from the host computer 4. In yet  
15 another embodiment, the disk 8 comprises a plurality of tracks, where each track comprises a  
16 plurality of sectors. The configuration parameters configure a number of sectors read into the  
17 semiconductor memory 26 during a read operation following a target sector of the read operation.  
18 In another embodiment, the configuration parameters configure a number of sectors read into the  
19 semiconductor memory 26 during a read operation preceding a target sector of the read  
20 operation.

21 In one embodiment, the host computer 4 of FIG. 1 comprises a computer readable storage  
22 medium for storing a computer program. The computer program for configuring the disk drive 2  
23 by transmitting setup data in a Self Monitoring Analysis and Reporting Technology (SMART)  
24 command 6 to the disk drive 2. The SMART command 6 comprises a command code 18, a sub  
25 command 20, and a buffer 22. The disk drive 2 comprises a disk 8, a head 10 actuated radially  
26 over the disk 8, an error recovery system 12 for detecting and correcting errors in user data read  
27 from the disk 8, and a cache system 14 for caching user data received from the host computer 4

1 and user data read from the disk 8. The disk drive 2 further comprises a plurality of configuration  
2 parameters stored in a non-volatile manner for configuring at least one system when the disk drive  
3 2 is powered on, the at least one system selected from the group consisting of the cache system  
4 14 and the error recovery system 12. The computer program comprises code segments for  
5 assigning a value to the command code 18 identifying the command code as a SMART command,  
6 assigning a value to the sub command 20 identifying the sub command as a disk drive setup  
7 command, and assigning setup data to the buffer 22, the setup data for modifying the  
8 configuration parameters of the disk drive 2. The computer program further comprises a code  
9 segment for transmitting the SMART command 6 to the disk drive 2.

10 FIG. 3 shows an embodiment of the present invention wherein the computer program is a  
11 configuration utility comprising a graphical user interface for displaying a drive setup window 46.  
12 The drive setup window 46 comprises a plurality of controls manipulated by the user in order to  
13 optimize the performance of the disk drive 2. In one embodiment, the controls include a check  
14 box 48 for auto-optimizing the disk drive 2 relative to the system configuration of the host  
15 computer 4. The auto-optimize function in one embodiment scans the disk drive 2 to determine  
16 the type of applications that the user may execute and configures the disk drive 2 accordingly.  
17 For example, if the auto-optimize function only finds Internet and multi-media applications, the  
18 disk drive 2 may be configured to reduce the correction power and associated latency of the error  
19 recovery system 12. In another embodiment, the auto-optimize function may configure the cache  
20 system 14 of the disk drive 2 relative to the caching configuration of the host computer 4. For  
21 example, the host computer 4 may allocate a portion of its internal semiconductor memory for use  
22 in caching data associated with the disk drive 2. In yet another embodiment, the host computer 4  
23 may provide error recovery functions which would allow the auto-optimize function to reduce the  
24 correction power and associated latency of the error recovery system 12 in the disk drive 2.

25 The controls in the embodiment of FIG. 3 further comprise a plurality of check boxes 50  
26 associated with various types of applications, such as word processing, page layout, image  
27 processing, CAD, Internet, data base, and multi-media. The user may configure the disk drive 2

1 by selecting the applications that will be run on the host computer 4. For example, if the user  
2 selects audio/video applications, such as Internet and multi-media, then the configuration utility  
3 will configure the error recovery system 12 of the disk drive 2 to reduce the correction power and  
4 associated latency. If the user selects applications having more critical data, such as word  
5 processing and data base applications, the configuration utility will configure the error recovery  
6 system 12 to increase the correction power such as enabling the parity sector mode described  
7 above.

8 The drive setup window 46 of FIG. 3 also comprises a control 52 for manually configuring  
9 the correction power and associated latency of the error recovery system 12. The user can adjust  
10 the correction power for better or worse reliability by adjusting the tab 54 on a slide bar. The  
11 performance of the disk drive 2 decreases as the user increases the reliability and associated  
12 latency of the error recovery system 12. The drive setup window 46 comprises a bar graph 56  
13 showing how the performance of the disk drive 2 is modified as the user manipulates the controls.

14 FIG. 4 shows an alternative embodiment of the present invention wherein the host  
15 computer is an Internet web server 58 which executes the configuration utility via the Internet 60.  
16 The end user communicates with the web server 58 via an Internet browser program executed by  
17 a computer 62. A drive setup page is downloaded from the web server 58 and displayed in a  
18 drive setup window 46 of the Internet browser program. As the user manipulates the controls in  
19 the drive setup window 46, the user input is transmitted back to the web server 58 via the Internet  
20 60. The web server evaluates the user input and generates the appropriate SMART command,  
21 including the drive setup data. The SMART command is then transmitted over the Internet 60 to  
22 computer 62 which forwards the SMART command to the disk drive 2. The web server 58  
23 implements suitable software, such as Java scripts, to effectuate the SMART command transfer to  
24 the disk drive 2 via the Internet 60 and computer 62.

25 In one embodiment, the disk drive 2 is configured by a user application program running  
26 on the host computer 4. For example, a word processing or accounting program may configure  
27 the disk drive 2 to increase the correction power of the error recovery system 12. The

1 configuration may take place when the application is launched, or the application may configure  
2 the disk drive 2 dynamically each time it performs an operation, such as a write operation. For  
3 example, each time an accounting program performs a write operation, it may first enable a parity  
4 mode of the disk drive 4 so that the data written is afforded additional protection. Once the write  
5 operation terminates, the accounting program may reconfigure the disk drive 2 to its state prior to  
6 the write operation. In this manner, each active user application program can dynamically  
7 configure the disk drive 2 to achieve the desired reliability, performance, or other configuration  
8 metric.

PATENT ATTY DOCKET K35A0840

SOURCE CODE APPENDIX

```
1 //=====
2 // FILE:      SMARTAPI.CPP
3 // DESCRIPTION: This file contains functions that allow a
4 //               Windows application to transfer reserved area files,
5 //               and modify individual bits, bytes or words within the
6 //               config sector. This is accomplished by utilizing the
7 //               DeviceIOControl windows API function that calls the
8 //               device driver SMARTVSD.VXD. Because there is additional
9 //               functionality that is not WD specific (such as executing
10 //              a IDENTIFY DRIVE command), there are additional functions
11 //              added to provide functionality for as much of what
12 //              SMARTVSD.VXD provides as possible
13 //
14 //
15 //
16 //
17 //=====
```

```
18 typedef struct _IDEREGS {
19     BYTE          chFeaturesReg;        // Used for specifying DFP "sub commands".
20     BYTE          chSectorCountReg;    // IDE sector count register
21     BYTE          chSectorNumberReg;   // IDE sector number register
22     BYTE          chCylLowReg;         // IDE low order cylinder value
23     BYTE          chCylHighReg;        // IDE high order cylinder value
24     BYTE          chDriveHeadReg;      // IDE drive/head register
25     BYTE          chCommandReg;       // Actual IDE command. Checked for validity by driver.
26     BYTE          chReserved;         // reserved for future use. Must be zero.
27 } IDEREGS, *PIDEREGS, *LPIDEREGS;
28
29 typedef struct _SENDCMDINPARAMS {
30     DWORD         dwBufferSize;       // Size of bBuffer in bytes
31     IDEREGS      irDriveRegs;        // Structure with drive register values.
32     BYTE          chDriveNumber;      // Physical drive number to send command to (0,1,2,3).
33     BYTE          chReserved[3];       // Reserved for future expansion.
34     DWORD         dwReserved[4];       // Reserved for future expansion.
35     BYTE          chBuffer[1];        // Buffer of arbitrary length in which to store the data to be written to drive.
36 } SENDCMDINPARAMS, *PSENDCMDINPARAMS, *LPSENDCMDINPARAMS;
37
38 typedef struct SendCmdOutParams {
39     DWORD         dwBufferSize;       // Size of bBuffer in bytes
40     DRIVERSTATUS  DriverStatus;       // Driver status structure.
41     BYTE          chBuffer[1];        // Buffer of arbitrary length in which to store the data read from the drive.
42 } SENDCMDOUTPARAMS, *PSENDCMDOUTPARAMS, *LPSENDCMDOUTPARAMS;
43
44 typedef struct {
45     WORD          wActionCodeAndWordOffset;
46     BYTE          chLSDataByteOrMask;
47     BYTE          chMSDDataByteOrMask;
48 } MODIFY_BLOCK, *PMODIFY_BLOCK;
49
50 typedef struct {
51     BYTE          chActionCode;
52     BYTE          chKeyFormat;
```

```
1      BYTE          chFileIDNumber;
2      BYTE          chAdditionalDataSectors;
3      CHAR          sSignature[12];
4      WORD          wModifyBlockCount;
5      MODIFY_BLOCK  aModifyBlock[123];
6      CHAR          chReserved;
7      BYTE          chChecksum;
8 } MODIFYBLOCK_KEYSECTOR, *PMODIFYBLOCK_KEYSECTOR;
9
10 // These go inside the Task File for Write and Read Logging Sector Commands
11 //
12 const BYTE kchWriteLoggingSectorFeaturesRegister = 0xD6;
13 const BYTE kchReadLoggingSectorFeaturesRegister = 0xD5;
14 const BYTE kchDoIdentifyFeaturesRegister = 0x00;
15 const BYTE kchAnyKeySectorSectorCountRegister = 0x01;
16 const BYTE kchGenericSmartSectorCountRegister = 0x01;
17 const BYTE kchAnyKeySectorSectorNumberRegister = 0xBE;
18 const BYTE kchGenericSmartSectorNumberRegister = 0x01;
19 const BYTE kchReadOrWriteReservedAreaFileSectorNumberRegister = 0xBF;
20 const BYTE kchAnyLoggingSectorCylLowRegister = 0x4F;
21 const BYTE kchDoIdentifyCylLowRegister = 0;
22 const BYTE kchDoIdentifyCylHighRegister = 0;
23 const BYTE kchAnyLoggingSectorCylHighRegister = 0xC2;
24 const BYTE kchAnyLoggingSectorCommandRegister = 0xB0;
25
26 // These go inside the Key Sector Data area (512 bytes)
27 //
28 const BYTE kchTransferMode = 0x81;
29 const BYTE kchReadReservedKeySectorActionCode = 0x01;
30 const BYTE kchWriteReservedKeySectorActionCode = 0x04;
31 const BYTE kchModifyBlockKeySectorActionCode = 0x05;
32 const BYTE kchModifyBlockKeySectorKeyFormat = 0x01;
33 const BYTE kchModifyBlockFileIDNumber = 0x42;
34 const BYTE ksReadReservedFileSignature[] = "WDC^F$&p#\\"; // Note one \ is an escape for the other backslash
35 const BYTE ksWriteReservedFileSignature[] = "WDC^S#p*/i$(";
36 const BYTE ksModifyBlockSignature[] = "WDC^M)s&!U:+";
37
38 =====
39 // Name:      OpenSMART
40 //
41 // Description: This function will perform operating system dependent
42 //               calls that open a handle to the SMART API. This handle
43 //               is used by nearly every function in the file to
44 //               access SMART API pass-through calls. Note when the
45 //               caller of this function cause a handle to be opened,
46 //               some function on the call stack must eventually call
47 //               CloseHandle.
48 //
49 =====
50
51 HANDLE OpenSMART( // return the result of CreateFile
52     VOID)        // VOID parameter
53 {
54     HANDLE      hSMARTIOCTL = 0;
```

```
1 #ifdef WINDOWS9X
2 // Version Windows 95 OSR2, Windows 98
3 hSMARTIOCTL = CreateFile("\\\\.\\"SMARTVSD",
4 0,
5 0,
6 0,
7 0,
8 CREATE_NEW,
9 0,
10 0);
11 #else
12 // Windows NT, Windows 2000
13 hSMARTIOCTL = CreateFile("\\\\.\\"PhysicalDrive0",
14 GENERIC_READ | GENERIC_WRITE,
15 FILE_SHARE_READ|FILE_SHARE_WRITE,
16 NULL,
17 OPEN_EXISTING,
18 0,
19 NULL);
20 #endif
21
22 return hSMARTIOCTL;
23
24 }
25
26
27 //=====
28 // Name: DoEnableSMART
29 //
30 // Description: This function will enable the target drive to accept
31 // SMART commands.
32 //
33 //=====
34
35 BOOL DoEnableSMART( // return the result of DeviceIOControl
36     HANDLE hSMARTIOCTL, // Handle returned from previous call to OpenSMART,
37     PSENDCMDINPARAMS pSCIP, // Structure that describes command details
38     PSENDCMDOUTPARAMS pSCOP, // Structure that DeviceIOControl fills with results
39     BYTE chDriveNum, // DriveNum = 0 through 3
40     PDWORD pdwBytesReturned) // Number of Bytes Returned From DeviceIOControl
41 {
42
43     BOOL bRetVal = TRUE; // assume success for case where DO_IO_CONTROL is not defined
44     //
45     // Set up data structures for Enable SMART Command.
46     //
47     pSCIP->dwBufferSize = 0;
48
49     pSCIP->irDriveRegs.chFeaturesReg = SMART_ENABLE_SMART_OPERATIONS;
50     pSCIP->irDriveRegs.chSectorCountReg = kchGenericSmartSectorCountRegister;
51     pSCIP->irDriveRegs.chSectorNumberReg = kchGenericSmartSectorNumberRegister;
52     pSCIP->irDriveRegs.chCylLowReg = kchAnyLoggingSectorCylLowRegister;
53     pSCIP->irDriveRegs.chCylHighReg = kchAnyLoggingSectorCylHighRegister;
54 }
```

```
1 //  
2 // Compute the drive number.  
3 //  
4 pSCIP->irDriveRegs.chDriveHeadReg = kchAnyLoggingSectorDriveHeadObsoleteBits | ((chDriveNum & 1) << 4);  
5 pSCIP->irDriveRegs.chCommandReg = kchAnyLoggingSectorCommandRegister;  
6 pSCIP->chDriveNumber = chDriveNum;  
7  
8 #if defined(DO_IO_CONTROL)  
9     bRetVal = ( DeviceIoControl(hSMARTIOCTL, DFP_SEND_DRIVE_COMMAND,  
10             (LPVOID)pSCIP, sizeof(SENDCMDINPARAMS) - 1,  
11             (LPVOID)pSCOP, sizeof(SENDCMDOUTPARAMS) - 1,  
12             pdwBytesReturned, NULL) );  
13 #endif  
14  
15     return bRetVal;  
16 }  
17  
18 //=====  
19 // Name:    DoModifyBlockKeySector  
20 //  
21 //  
22 // Description: This function writes a sector that allows you to set  
23 //      the value of a byte, word, or bit in the config  
24 //      sector. You can specify 123 such "modifications"  
25 //      by setting up Modify Block data within the key sector.  
26 //      Note that you cannot change the first 24  
27 //      bytes of the config sector with this command. Also  
28 //      note that although there are components of the  
29 //      SMART firmware that support two sectors of modify  
30 //      block data, WD Firmware Engineering (see Mark Vallis)  
31 //      has determined that we only need to implement one  
32 //      sectors worth of modify block data.  
33 //  
34 //      This function performs the write  
35 //      by calling the DeviceIOControl function that in turn  
36 //      calls the SMARTVSD.VXD (or equivalent) driver. The  
37 //      key sector data that is written in a buffer that is  
38 //      part of the PSENDCMDINPARAMS structure. As a result,  
39 //      the data MUST be adjacent in memory with the  
40 //      rest of the PSENDCMDINPARAMS structure. Note that  
41 //      the caller of this function MUST allocate extra space  
42 //      for the data written in the PSENDCMDINPARAMS  
43 //      structure before calling this function.  
44 //=====  
45 BOOL DoModifyBlockKeySector( // return the result of DeviceIOControl  
46     HANDLE hSMARTIOCTL, // Handle returned from previous call to OpenSMART  
47     PSENDCMDINPARAMS pSCIP, // Structure that describes command details (including task file input AND output)  
48     PSENDCMDOUTPARAMS pSCOP, // Structure that DeviceIOControl fills with results, also contains read buffer  
49     BYTE chDriveNum, // DriveNum = 0 through 3  
50     PMODIFY_BLOCK pModifyBlock, // Array of MODIFY_BLOCK structs that the key sector will be filled with  
51     WORD wNumModifyBlocks, // Number of Valid MODIFY_BLOCKS in the previous parameter  
52     PDWORD pdwBytesReturned) // Number of Bytes Returned From DeviceIOControl  
53 {  
54 }
```

```
1  BOOL bRetVal = 1;
2
3 // Note: The first backslash is for the escape character on ksReadReservedFileSignature
4 //
5
6 // Point to the KeySector part of the KeyCmd
7 //
8 MODIFYBLOCK_KEYSECTOR KeySector;
9 memset((void*) &KeySector, 0, sizeof(KeySector));
10
11 //
12 // Set up data structures for DFP_SEND_DRIVE_COMMAND command.
13 //
14 pSCIP->dwBufferSize = kwKEY_SECTOR_BUFFER_SIZE; // - sizeof(SENDcmdINPARAMS) + 1;
15
16 // Note that when DeviceIOControl has returned these input parameters are
17 // filled with output values from the drive.
18 //
19 pSCIP->irDriveRegs.chFeaturesReg = kchWriteLoggingSectorFeaturesRegister;
20 pSCIP->irDriveRegs.chSectorCountReg = kchAnyKeySectorSectorCountRegister;
21 pSCIP->irDriveRegs.chSectorNumberReg = kchAnyKeySectorSectorNumberRegister;
22 pSCIP->irDriveRegs.chCylLowReg = kchAnyLoggingSectorCylLowRegister;
23 pSCIP->irDriveRegs.chCylHighReg = kchAnyLoggingSectorCylHighRegister;
24 pSCIP->irDriveRegs.chDriveHeadReg = kchAnyLoggingSectorDriveHeadObsoleteBits | ((chDriveNum & 1) << 4);
25 pSCIP->irDriveRegs.chCommandReg = kchAnyLoggingSectorCommandRegister;
26 pSCIP->chDriveNumber = chDriveNum;
27
28 KeySector.chActionCode = kchModifyBlockKeySectorActionCode;
29 KeySector.chKeyFormat = kchModifyBlockKeySectorKeyFormat;
30 KeySector.chFileIDNumber = kchModifyBlockFileIDNumber;
31 memcpy(KeySector.sSignature, ksModifyBlockSignature, 12);
32 KeySector.chAdditionalDataSectors = 0;
33 KeySector.wModifyBlockCount = wNumModifyBlocks;
34
35 // Copy the ModifyBlock data into the KeySector
36 //
37 memcpy(KeySector.aModifyBlock, pModifyBlock, wNumModifyBlocks * sizeof(MODIFY_BLOCK));
38
39 // Copy the KeySector into the packet buffer
40 //
41 memcpy(pSCIP->chBuffer, &KeySector, kwKEY_SECTOR_BUFFER_SIZE);
42
43 memset(pSCOP, 0x00, sizeof(SENDcmdOUTPARAMS));
44
45 #if defined(DO_IO_CONTROL)
46 bRetVal = (DeviceIoControl(hSMARTIOCTL, DFP_SEND_DRIVE_COMMAND,
47 (LPVOID)pSCIP, sizeof(SENDcmdINPARAMS) + kwKEY_SECTOR_BUFFER_SIZE - 1,
48 (LPVOID)pSCOP, sizeof(SENDcmdOUTPARAMS) - 1,
49 pdwBytesReturned, NULL));
50#endif
51
52 return bRetVal;
53
54 }
```